WEST Search History

Hide Items Restore Clear Cancel

DATE: Thursday, March 11, 2004

Hide?	Set Name	Query	Hit Count
		DB=USPT,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR	
☐	L43	l21 same (shift adj register)	12
☐	L42	L40 same l21	3
☐	L41	L40 and l21	51
☐	L40	(entry near2 shift$4)	1232
☐	L39	l21 same (check$4 near3 flag)	5
☐	L38	l1 same (check$4 near2 depth)	0
☐	L37	L35.ab.	4
☐	L36	L35.clm.	1
☐	L35	L34 same flag	32
☐	L34	(process$4 adj (order or sequence)) near3 (request or command or instruction)	1255
☐	L33	(process$4 near3 (order or sequence) near3 (request or command or instruction))	8158
☐	L32	l29.clm.	6
☐	L31	l29.ab.	7
☐	L30	l17 and L29	11
☐	L29	L27 same (queue or fifo or buffer)	81
☐	L28	L27 same (queue or fifo or buffer or memory)	237
☐	L27	L26 same stor$4	419
☐	L26	(process$4 near3 (request or command or instruction)) same ((correspond$4 or associat$4 or relat$4) near3 (index or key or pointer))	799
☐	L25	(request or command or instruction) same ((correspond$4 or associat$4 or relat$4) near3 (index or key or pointer))	6977
☐	L24	l17 and L23	4
☐	L23	L22 same process$4	300
☐	L22	L21 same (correspond$4 or associat$4)	702
☐	L21	l19 same L20	2968
☐	L20	(stor$4 near3 (index or key or pointer))	52057
☐	L19	(stor$4 near3 (request or command or instruction))	103077
☐	L18	l5 and L17	13
☐	L17	l9 or l10 or l11 or l12 or l13 or l14 or l15 or L16	3376

☐	L16	710/5.ccls.	649
☐	L15	711/159.ccls.	256
☐	L14	711/109.ccls.	196
☐	L13	711/154.ccls.	943
☐	L12	710/310.ccls.	288
☐	L11	710/55.ccls.	93
☐	L10	710/54.ccls.	159
☐	L9	710/52.ccls.	1162
☐	L8	L6 same (key or index or pointer)	5
☐	L7	l1 and L6	0
☐	L6	((plurality or multiple) near2 (fixed or static) near2 registers)	58
☐	L5	l1.ab.	529
☐	L4	L3 same (key or index or pointer)	11
☐	L3	((queue or buffer or fifo) near2 entry near2 pool)	20
☐	L2	L1 same (entry near2 pool)	0
☐	L1	(queue or fifo or buffer) adj structure	2401

END OF SEARCH HISTORY

First Hit Fwd Refs

☐ Generate Collection Print

L43: Entry 4 of 12 File: USPT Mar 2, 1982

DOCUMENT-IDENTIFIER: US 4318172 A
TITLE: Store data buffer control system

Detailed Description Text (5):
The shift register 201 of the re-execution control 20 is of a four-bit type and serves to take in storing requests delivered from the instruction control 10 and sent through a signal line 100 and to trace those storing requests which are executable in the instruction re-execution. The processor 1 supplies shift pulses to the shift register 201 in timing with successive shift cycles so that only the storing requests delivered in the previous four cycles may be preserved in the register 201. For example, in the (n+3)th cycle, the 4-bit shift register 201 traces the states of storing requests delivered in the last four cycles, i.e. n-th to (n+3)th cycles. The flip-flops 201a-201d of the shift register 201 indicates the states of the (n+3)th to the n-th cycles, respectively. For example, if store requests are generated in the n-th and the (n+2)th cycles, the shift register 201, as seen in FIG. 3, will have a content "0101". Each of the flip-flops 201a to 201d has a data input terminal D and a trigger terminal T for receiving shift pulses. The decoder 202 receives as inputs the contents of the shift register 201 and the buffer input pointer 302 and delivers a storing request nullifying instruction signal to the store data buffer control 30 at the time of instruction re-execution after the occurrence of a fault. The control 203 receives a signal indicating the occurrence of a fault from the processor 1 via a line 102 at the time of the fault occurring and performs a control necessary for the re-execution of instructions, that is, delivers a reset signal to a signal line 204 and a storing request re-starting instruction signal to a signal line 205.

Detailed Description Text (8):
The storing request signal generated by the instruction control 10 is also received by the shift register 201 of the re-execution control 20 through the signal line 100. The storing request signal received by the shift register 201 is sequentially shifted rightward from bit position 201a toward bit position 201d in response to shift pulses sent through the line 101 from the processor 1 and vanishes every four shift cycles. Namely, the content of the shift register 201 represents the successive storing requests delivered in the four previous sequential cycles. At the time of instruction re-execution after the occurrence of a fault, the control 203 delivers a reset instruction signal to a signal line 204 and a storing request restarting instruction signal to a signal line 205. When the decoder 202 receives the reset instruction signal, it generates a storing request nullifying instruction signal to nullify the content of the store data buffer corresponding to the storing request remaining then in the shift register 201. For example, if two storing requests remain in the shift register 201 and if the content of the buffer input pointer 302 represents a value "3", then the store data buffers 4-1 and 4-2 are selected in the instruction re-execution. The decoder 202 receives as inputs the contents of the shift register 201 and the buffer input pointer 302 and generates a storing request nullifying instruction signal to nullify the contents of the store data buffers specified in the instruction re-execution. The storing request nullifying instruction signal generated by the decoder 202 is used through the selection circuit 306 to reset the store data representing flip-flops 301-1 to 301-4. As a result of resetting the flip-flops, the contents of the corresponding store data buffers are nullified so that the storing requests are re-executed under the

control operation of the re-execution control 20 at the time of instruction re-execution. The control 308 of the store data buffer control 30 receives a storing request re-starting instruction signal from the re-execution control 20 through the signal line 205 and if there then remains a store data representing flip-flop which is still in the set state without having been reset in the previous processing of nullifying the contents of the store data buffers, the control 308 delivers a new storing request signal to a signal line 309 so as to re-execute only storing operations.

First Hit Fwd Refs

☐ Generate Collection Print

L30: Entry 3 of 11 File: USPT May 20, 2003

DOCUMENT-IDENTIFIER: US 6567901 B1
TITLE: Read around speculative load

Detailed Description Text (19):
This is a logical description of the memory request insertion process; a system according to the invention could instead prioritize the memory requests in many different ways. For example, it can insert the memory request entry into any available spot in the queue and then resort the entire queue to maintain the relative ordering without changing the spirit of the invention. More preferably, the method of the disclosed technique can use pointers to memory requests and their associated statuses stored in another storage area so that, during a sort, only the pointers are changed and the memory requests and the statuses need not be moved. In addition, another embodiment can have separate queues for the processing of speculative and non-speculative memory requests. The memory controller 14 can process all requests in the non-speculative queue before servicing any requests from the speculative queue. Similarly, separate queues are often provided for code and data--these, too, can have associated speculative/non-speculative status or separate speculative/non-speculative queues. A further embodiment which uses a "needed, but not right now" status can employ a third queue.

Current US Cross Reference Classification (1):
711/154

First Hit Fwd Refs

☐ Generate Collection Print

L30: Entry 3 of 11 File: USPT May 20, 2003

DOCUMENT-IDENTIFIER: US 6567901 B1
TITLE: Read around speculative load

Detailed Description Text (19):
This is a logical description of the memory request insertion process; a system according to the invention could instead prioritize the memory requests in many different ways. For example, it can insert the memory request entry into any available spot in the queue and then resort the entire queue to maintain the relative ordering without changing the spirit of the invention. More preferably, the method of the disclosed technique can use pointers to memory requests and their associated statuses stored in another storage area so that, during a sort, only the pointers are changed and the memory requests and the statuses need not be moved. In addition, another embodiment can have separate queues for the processing of speculative and non-speculative memory requests. The memory controller 14 can process all requests in the non-speculative queue before servicing any requests from the speculative queue. Similarly, separate queues are often provided for code and data--these, too, can have associated speculative/non-speculative status or separate speculative/non-speculative queues. A further embodiment which uses a "needed, but not right now" status can employ a third queue.

Current US Cross Reference Classification (1):
711/154

First Hit Fwd Refs

☐ Generate Collection Print

L30: Entry 5 of 11 File: USPT Mar 26, 2002

DOCUMENT-IDENTIFIER: US 6363438 B1
TITLE: Method of controlling DMA command buffer for holding sequence of DMA commands with head and tail pointers

Abstract Text (1):
A direct memory access (DMA) controller is provided for a computer system having a processor and a command buffer. The command buffer can be defined, for example, as a ring buffer in the main processor memory and can be directly accessible by the processor, for example over a bus. The DMA controller provides a head register and a tail register operable to hold a head pointer and a tail pointer for addressing the head and tail, respectively, of a sequence of direct memory access commands in the command buffer. The processor is able to store DMA commands in the command buffer. Subsequently, the DMA controller is able to access those DMA commands using the DMA tail pointer held locally in the DMA controller. The DMA controller is operable to compare the head and tail pointers, and to respond to non-equivalence thereof to use the tail pointer value to access direct memory access commands from the command buffer. The DMA controller is responsible for updating the tail pointer in the DMA controller in association with reading of a direct memory access command from a location in the command buffer. The processor is responsible for updating the head pointer in the DMA controller in association with the storing of DMA commands in the command buffer.

Current US Cross Reference Classification (5):
710/52

First Hit Fwd Refs

☐ Generate Collection Print

L31: Entry 1 of 7 File: USPT Mar 26, 2002

DOCUMENT-IDENTIFIER: US 6363438 B1
TITLE: Method of controlling DMA command buffer for holding sequence of DMA commands with head and tail pointers

Abstract Text (1):
A direct memory access (DMA) controller is provided for a computer system having a processor and a command buffer. The command buffer can be defined, for example, as a ring buffer in the main processor memory and can be directly accessible by the processor, for example over a bus. The DMA controller provides a head register and a tail register operable to hold a head pointer and a tail pointer for addressing the head and tail, respectively, of a sequence of direct memory access commands in the command buffer. The processor is able to store DMA commands in the command buffer. Subsequently, the DMA controller is able to access those DMA commands using the DMA tail pointer held locally in the DMA controller. The DMA controller is operable to compare the head and tail pointers, and to respond to non-equivalence thereof to use the tail pointer value to access direct memory access commands from the command buffer. The DMA controller is responsible for updating the tail pointer in the DMA controller in association with reading of a direct memory access command from a location in the command buffer. The processor is responsible for updating the head pointer in the DMA controller in association with the storing of DMA commands in the command buffer.

First Hit Fwd Refs

☐ Generate Collection Print

L32: Entry 1 of 6 File: USPT Jan 13, 2004

DOCUMENT-IDENTIFIER: US 6678704 B1
TITLE: Method and system for controlling recovery downtime by maintaining a checkpoint value

CLAIMS:

15. A computer-readable medium carrying one or more sequences of instructions for maintaining a checkpoint value that indicates which records of a plurality of records associated with updates made before a failure have to be processed after the failure, wherein execution of the one or more sequences of instructions by one or more processors causes the one or more processors to perform the steps of: maintaining, in volatile memory, a sorted buffer queue that includes a head and a tail, wherein the sorted buffer queue includes queue entries that are inserted into said sorted buffer queue based on an index value that is associated with each queue entry; removing queue entries from said sorted buffer queue only after information associated with said queue entries is stored in nonvolatile memory; and periodically updating the checkpoint value to equal the index value that is associated with the queue entry that is currently at the head of the sorted buffer queue.

First Hit Fwd Refs

☐ Generate Collection Print

L32: Entry 2 of 6 File: USPT Oct 14, 2003

DOCUMENT-IDENTIFIER: US 6633575 B1
TITLE: Method and apparatus for avoiding packet reordering in multiple-class, multiple-priority networks using a queue

CLAIMS:

28. An article of manufacture for packet queuing without packet reordering, the article of manufacture comprising a computer readable medium having instructions for causing a processor to perform a method comprising: receiving packets in a first queue, the first queue being associated with a first traffic class, the packets in the first queue comprising doubly-linking; arranging the received packets in the first queue as in-profile packets and out-profile packets; and storing pointers associated with the out-profile packets in a second queue, the pointers in the second queue being associated with corresponding out-profile packets in the first queue and to a previous and next element in the second queue.

First Hit Fwd Refs
End of Result Set

☐ Generate Collection Print

```
L36: Entry 1 of 1                    File: USPT                    Jul 6, 1993
```

DOCUMENT-IDENTIFIER: US 5226112 A
TITLE: Method for translating a plurality of printer page description languages

CLAIMS:

8. A method for swapping portions of a first memory map having a sequence of instructions and corresponding addresses with portions of a second memory map having a sequence of instructions and corresponding addresses, the first memory map being disposed in the processing means and portions of the second memory map being stored in a storage section, the first and second memory maps being operatively associated with first and second interrupt routines, respectively, each of the interrupt routines having a sequence of instructions with corresponding addresses, comprising the steps of:

swapping portions of the first memory map with the portions of the second memory map in accordance with a swapping routine having a sequence of instructions with corresponding addresses, one of the addresses in the sequence of instructions for the swapping routine representing a flag, the flag being set at a selected one of a first status and a second status;

processing the sequence of instructions for the first memory map;

interrupting, at selected intervals, the processing step to check the status of the flag in the sequence of instructions for the swapping routine;

<u>processing the sequence of instructions</u> in the first interrupt routine when the <u>flag</u> is set at the first status and, when the <u>flag</u> is set at the second status, <u>processing the sequence of instructions</u> in the swapping routine so that the portions of the first memory map are stored in the storage section and portions of the second memory map are disposed in the processing means.

First Hit Fwd Refs

☐ Generate Collection Print

L37: Entry 1 of 4 File: USPT Oct 31, 2000

DOCUMENT-IDENTIFIER: US 6141734 A
TITLE: Method and apparatus for optimizing the performance of LDxL and STxC interlock instructions in the context of a write invalidate protocol

Abstract Text (1):
A technique for implementing load-locked and store-conditional instruction primitives by using a local cache for information about exclusive ownership. The valid bit in particular provides information to properly execute load-locked and store-conditional instructions without the need for lock flag or local lock address registers for each individual locked address. Integrity of locked data is accomplished by insuring that load-locked and store-conditional instructions are processed in order, that no internal agents can evict blocks from a local cache as a side effect as their processing, that external agents update the context of cache memories first using invalidating probe commands, and that only non-speculative instructions are permitted to generate external commands.

First Hit Fwd Refs

☐ Generate Collection Print

L39: Entry 1 of 5 File: USPT Feb 16, 1999

DOCUMENT-IDENTIFIER: US 5872985 A
TITLE: Switching multi-context processor and method overcoming pipeline vacancies

Detailed Description Text (19):
FIG. 9 is a functional block diagram for realizing the multi-register renaming of the invention and shows the details of the multi-register renaming section 35 in FIG. 4. The multi-register renaming section 35 is constructed by using the CPU space 10-1 and memory space 16-1. A context mapping table 62 is provided for the CPU space 10-1. In case of starting the execution of an almost new context, a context number CN is registered into the context mapping table 62 by a support of the OS. Since the embodiment relates to the example in the case where there are four contexts which are simultaneously run, context IDs 00, 01, 10, and 11 are allocated to the context mapping table 62. For this context ID, #0 is registered into the context ID00 as a context number CN0 as an execution target. A context No. #1 is registered into the context ID10. Further, an in-execution context ID register 76, the instruction register 78, a renaming buffer 82, and a save/load processing section 92 are provided for the CPU space 10-1. The context which is at present being executed, for example, the context ID=10 of the context No. #1 is stored in the in-execution context ID register 76. An instruction code in which the register name as a renaming target has been stored in a register designation field 80 is stored in the instruction register 78. The renaming buffer 82 is divided to register areas of physical register sections 84-1 to 84-16 which can be designated by the register designation field 80 of the instruction. In the embodiment, although the number of physical registers which can be designated has been set to 16, it can be set to a proper number such as 32, 64, or the like. Each of the physical register sections 84-1 to 84-16 of the renaming buffer 82 has a valid flag field 86, a key field 88, and a data field 90. In the valid flag field 86, the valid flag is turned on when the physical register is used as a renaming register. When the physical register is not used, the valid flag is turned off. Therefore, by checking the valid flag 86 of each of the physical register sections 84-1 to 84-16, the presence or absence of the physical registers which can be used for renaming can be judged. A key in which CIDi stored in the in-execution context ID register 76 and the register name Rj of the register designation field 80 of the instruction register 78 are combined is stored in the key field 88 as a key code indicative of the renaming register name. Therefore, even in case of the same register name Rj, the registers can be distinguished by CIDi as a context ID. On the other hand, context control blocks 94-1 and 94-2 are assured in the memory space 16-1 every context numbers #0 and #1 registered in the context mapping table 62. In the context control blocks 94-1 and 94-2, specific areas are allocated to register saving areas 98-1 and 98-2. As register saving areas 98-1 and 98-2, the areas of the same number as that of the physical register sections 84-1 and 84-16 of the renaming buffer 82 are fundamentally assured. Each register area of the register saving area 98-1 is divided to a valid flag field 100 and a register data field 102. When the register data is saved to the register data field 102, a valid flag of the valid flag field 100 is turned on. When the register data is loaded to the renaming buffer 82, the valid flag is turned off. In case of the register saving area 98-1 of the context No. #0, only the head valid flag field 100 is set to 0 and the other valid flags are set to 1. This means that only the head register data exists in the renaming buffer 82. On the other hand, with respect to the register saving area 98-2 of the context No. #1 of the context which is at present being

executed, all of the valid flags are equal to 0 and are OFF. This means that all of the renaming registers exist in the renaming buffer 82. The areas of the context control blocks 94-1 and 94-2 other than the register saving areas 98-1 and 98-2 are used as ordinary control areas 110-1 and 110-2. The save/load processing section 92 executes a register saving process for the memory space 16-1 when the renaming buffer 82 overflows and the loading process of the register data from the memory space 16-1 when the renaming register which is used in the context that is being executed doesn't exist in the renaming buffer 82.

First Hit Fwd Refs

☐ Generate Collection Print

L42: Entry 1 of 3 File: USPT Sep 30, 2003

DOCUMENT-IDENTIFIER: US 6629218 B2
TITLE: Out of order associative queue in two clock domains

Abstract Text (1):
A memory controller may include a request queue for receiving transaction information (e.g. the address of the transaction) and a channel control circuit. A control circuit for the request queue may issue addresses from the request queue to the channel control circuit out of order, and thus the memory operations may be completed out of order. The request queue may shift entries corresponding to transactions younger than a completing transaction to delete the completing transaction's information from the request queue. However, a data buffer for storing the data corresponding to transactions may not be shifted. Each queue entry in the request queue may store a data buffer pointer indicative of the data buffer entry assigned to the corresponding transaction. The data buffer pointer may be used to communicate between the channel control circuit, the request queue, and the control circuit. In one implementation, the request queue may implement associative comparisons of information in each queue entry (e.g. transaction IDs and/or data buffer pointers). In one embodiment, the request queue and control circuit may be in the bus clock domain, while the channel control circuit may be in the memory clock domain.

Brief Summary Text (9):
The problems outlined above are in large part solved by a memory controller as described herein. The memory controller may include a request queue for receiving transaction information (e.g. the address of the transaction) and a channel control circuit which controls a memory bus. A control circuit for the request queue may issue addresses from the request queue to the channel control circuit out of order, and thus the memory operations may be completed out of order. In one embodiment, the request queue shifts entries corresponding to transactions younger than a completing transaction to delete the completing transaction's information from the request queue. However, a data buffer for storing the data corresponding to transactions may not be shifted. Each queue entry in the request queue may store a data buffer pointer indicative of the data buffer entry assigned to the corresponding transaction. The data buffer pointer remains constant throughout the life of the transaction in the memory controller, and may be used to communicate between the channel control circuit, the request queue, and the control circuit.

Detailed Description Text (22):
Data buffer 52 may not shift entries as transactions are completed. Accordingly, each queue entry of request queue 40 stores a data buffer pointer indicative of the data buffer entry assigned to the transaction corresponding to that queue entry (the Ptr field illustrated in entries 54A-54B). Control circuit 50 maintains a free list 58 of data buffer entries which are not currently in use for transactions represented in request queue 40, and assigns a data buffer entry from the free list for a memory transaction received on bus 24. The data buffer pointer corresponding to the assigned data buffer entry is stored in the assigned queue entry. The assigned data buffer entry is deleted from free list 58, and is added back to free list 58 when the transaction completes and is deleted from request queue 40 and data buffer 52.

First Hit Fwd Refs

End of Result Set

☐ Generate Collection Print

L24: Entry 4 of 4 File: USPT Mar 4, 1986

DOCUMENT-IDENTIFIER: US 4574349 A
TITLE: Apparatus for addressing a larger number of instruction addressable central processor registers than can be identified by a program instruction

Abstract Text (1):
Each of a plurality of stored pointers identifies and accesses one of a plurality of hardware registers in a central processing unit (CPU). Each pointer is associated with and corresponds to one of a limited number of general purpose registers addressable by various fields in a program instruction of the data processing system. At least one program instruction calls for transfer of data from a particular main storage location to a general purpose register (GPR) identified by a field in the program instruction. The GPR identified as the destination for the data is renamed by assigning a pointer value to provide access to one of the plurality of associated hardware registers. A subsequent load instruction involving the same particular main storage location determines if the data from the previous load instruction is still stored in one of the hardware registers and determines the associated pointer value. The data in the hardware register is made immediately available to the CPU before completion of the access to main storage. The pointer value is associated with, and made to correspond to the destination GPR of the subsequent load instruction. Other instructions which require access to instruction addressable GPR's cause access to the corresponding pointer value to provide access to the corresponding hardware register for purposes of data processing.

Current US Original Classification (1):
711/154

First Hit Fwd Refs

☐ Generate Collection Print

L30: Entry 1 of 11 File: USPT Oct 14, 2003

DOCUMENT-IDENTIFIER: US 6633575 B1
TITLE: Method and apparatus for avoiding packet reordering in multiple-class, multiple-priority networks using a queue

Current US Cross Reference Classification (2):
710/54

CLAIMS:

28. An article of manufacture for packet queuing without packet reordering, the article of manufacture comprising a computer readable medium having instructions for causing a processor to perform a method comprising: receiving packets in a first queue, the first queue being associated with a first traffic class, the packets in the first queue comprising doubly-linking; arranging the received packets in the first queue as in-profile packets and out-profile packets; and storing pointers associated with the out-profile packets in a second queue, the pointers in the second queue being associated with corresponding out-profile packets in the first queue and to a previous and next element in the second queue.

First Hit Fwd Refs

☐ Generate Collection Print

L24: Entry 1 of 4 File: USPT Aug 20, 2002

DOCUMENT-IDENTIFIER: US 6438650 B1
TITLE: Method and apparatus for processing cache misses

Detailed Description Text (3):
The present invention provides a system and method for efficiently processing transaction requests ("requests") to a system or main memory. On a cache miss, a cache controller stores information characterizing the request (request information) in a buffer. The request information is also provided to a secondary miss system, which identifies a bus transaction to service the request. The secondary miss system provides a bus controller with a pointer to the stored request information, and the pointer is associated with the identified bus request. Providing a pointer to the request information rather than the information itself reduces the number of signals that are provided to the bus controller. The identified bus transaction may be a pending transaction triggered by an earlier cache miss to the same cache line targeted by the current request. In this case, the current request is mapped to the pending bus transaction, which processes requests for multiple cache misses, reducing the amount of traffic on the system bus.

Current US Cross Reference Classification (3):
711/154

First Hit Fwd Refs

☐ Generate Collection Print

L24: Entry 2 of 4 File: USPT Mar 12, 2002

DOCUMENT-IDENTIFIER: US 6356972 B1
TITLE: System and method for concurrently requesting input/output and memory address space while maintaining order of data sent and returned therefrom

Brief Summary Text (24):
A bus interface unit is preferably provided within the computer. The bus interface unit is configured between a processor bus, a peripheral bus, and a memory bus. The bus interface unit includes an in-order queue coupled to store an order in which a plurality of requests are dispatched from the processor bus to either the peripheral bus or the memory bus. A peripheral request queue is coupled to store peripheral addresses associated with a first set of the plurality of requests destined exclusively for the peripheral bus. A memory request queue is coupled to store memory addresses associated with a second set of the plurality of requests destined exclusively for the memory bus. A comparator may be included and coupled between a pointer associated with the in-order queue and a pointer associated with data queues. The comparator is configured to dispatch the peripheral data and the memory data across the processor bus commensurate with the order in which the plurality of earlier-dispatched requests were stored in the in-order queue. More specifically, the comparator determines the relative position of the pointer attributed to the in-order queue. Based on that position, the comparator determines the next data to be sent from a queue having data resulting from that request. Once a match to data is ascertained, based on where the pointer resides in the in-order queue, that data is then forwarded across the processor bus (either as read data to the processor or as write data from the processor). In this manner, the current status of the pointer and the entry numbers stored within the pointer establish proper ordering of data subsequently forwarded across the processor bus even though requests may be sent to target devices out-of-order from requests earlier sent across the processor bus. Instances in which the requests are sent out-of-order occur due to peripheral requests and memory requests being sent concurrently, where one type of request is not delayed based on the other. As an alternative to the comparator, more simplistic logic can be implemented merely to pull data from the respective memory or peripheral data queues based on the order of requests maintained within the in-order queue. Avoidance of the comparator assumes requests are issued in-order and maintained in-order within respective data queues.

Current US Original Classification (1):
710/310

First Hit Fwd Refs

☐ Generate Collection Print

L8: Entry 4 of 5 File: USPT Aug 22, 1995

DOCUMENT-IDENTIFIER: US 5444660 A
TITLE: Sequential access memory and its operation method

Brief Summary Text (8):
A static type row address pointer 2a sequentially applies a plurality of row selecting signals Qr1-Qrn to a plurality of row selecting lines 4 for sequentially selecting one row of memory cell array 1. Row address pointer 2a includes a plurality of static type registers 30 for sequentially shifting data (row selecting signals) in synchronization with input clock signals, and even-numbered inverter circuits 31 for feeding back output signal of the last stage to the first stage of the registers 30. A inverter circuit 31 plays a role of a buffer for driving interconnection capacitance.

Brief Summary Text (9):
A static type column address pointer 3a sequentially applies column selecting signals Qc1-Qcm to column selecting lines 5 for sequentially selecting one column of memory cell array 1. As in the case of row address pointer 2a, column address pointer 3a includes a plurality of static type registers 30 for sequentially shifting data in synchronization with the input clock signals, and even numbered inverter circuits 31 for feeding back the output signal of the last stage to the first stage of the registers 30. The inverter circuits 31 also plays a role of a buffer for driving interconnection capacitance.

Brief Summary Text (21):
As mentioned above, in the conventional SAM, for the purpose of stably holding data (selection signals) a static type address pointer constituted by a plurality of static type registers 30 was employed in both row address pointer 2a and column address pointer 3a. For this reason, many transistors were used, and the occupation area of the address pointer on a semiconductor chip was increased. This was obstruction in obtaining an high integrated SAM.

First Hit Fwd Refs

☐ Generate Collection Print

L4: Entry 1 of 11 File: USPT Apr 15, 2003

DOCUMENT-IDENTIFIER: US 6549895 B1
TITLE: Method and apparatus for analyzing data retrieval using index scanning

Brief Summary Text (12):
In a traditional database management system, the database administrator or user can issue a request to collect statistics so that the system's optimizer has up-to-date information on tables and indexes. For example, DB2 Universal Database provides a command called "runstats" that allows users to collect indexes' clustering coefficients, among other statistics. DB2's runstats command also gives users the option of collecting FPF (i.e. Full index scan Page Fetch) information. Prior versions of DB2 (i.e. versions 6.1 and older) compute the FPF information by simulating a set of buffer pools while scanning an index. For each index entry, each simulated buffer pool is examined to determine if the index entry requires a data page transfer. The FPF information thus gathered is accurate, but the process of computation is extremely time-consuming. In contrast, the process of computing the clustering coefficient is very efficient, but can often yield poor estimates of the number of page transfers.

Detailed Description Text (21):
Referring next to FIG. 5, the method for the second processing operation 200 comprises the first sub-operation 200a and the second sub-operation 200b. As shown in FIG. 5, the first sub-operation 200a involves computing the proportion of index entries P.sub.I having a distance no less than I. This operation is implemented as a loop in blocks 201, 202 and 204. Once all the proportions have been computed for all the index entries I (decision block 203), the second sub-operation 200b is performed starting at block 211. The first step in block 211 involves initializing the variable D and the variable L. The variable D is an estimate of the average number of distinct page numbers in a group of L consecutive index entries. Next in block 212, the variables D and L are incremented until the variable D is greater than or equal to the buffer pool size B as determined in decision block 214. In this sense, the variable L becomes an estimate of the number of consecutive index entries that are required to fill the buffer pool (i.e. cache in main memory) having size B. After the number of consecutive index entries to fill the buffer pool is determined (block 214), an estimate of page transfers N*P.sub.L+1 is returned in block 216. The number of page transfers N*P.sub.L+1 is determined by considering that index entries having a distance greater than L will result in a page transfer.

First Hit Fwd Refs

☐ Generate Collection Print

L4: Entry 3 of 11 File: USPT Mar 5, 2002

DOCUMENT-IDENTIFIER: US 6353820 B1
TITLE: Method and system for using dynamically generated code to perform index record retrieval in certain circumstances in a relational database manager

Detailed Description Text (56):
The IO component layer 208 returns with a pointer to the fine level index CI that has been read into the buffer pool. RFM component layer 206 finds the next entry and sets up its currency information. It returns to the subroutine function RFM_IO8_GETNEXT with a pointer to the CI and a pointer to its currency information. The function RFM_IO8_GETNEXT then returns to the generated output code that analyzes the returned index key value. Because the second instance of `Jones` does not have `A` as a middle initial, the search continues. The generated output code again calls the subroutine function RFM_IO8_GETNEXT to retrieve the next index entry for `Jones`. Subroutine RFM_IO8_GETNEXT notes that it has all the necessary pointers to find the next fine level index entry in the index CI. Because the prior search did not produce a result, the generated output code did not set the result processed indicator. Therefore, the function RFM_IO8_GETNEXT uses its CI pointer and the RFM currency information to copy the next fine level index entry from the buffer pool to the generated output code key buffer.

First Hit Fwd Refs

☐ Generate Collection Print

L18: Entry 4 of 13 File: USPT Dec 3, 2002

DOCUMENT-IDENTIFIER: US 6490666 B1
TITLE: Buffering data in a hierarchical data storage environment

Abstract Text (1):
A system, a method, and program products for buffering data from a file in a hierarchical data storage system allocates data buffers and buffer management structures' in memory to optimize performance of no recall requests. Buffer management structures, such as buffer headers and hash queue headers, are used to optimize performance of insert, search, and data buffer reuse operations. Buffer headers are managed in a least-recently-used queue in accordance with a relative availability status. Buffer headers are also organized in hash queue structures in accordance with file-based identifiers to facilitate searching for requested data in the buffers. Data buffers can be used to buffer different data blocks within the same file and can be recycled to buffer data from other data blocks and other files from the secondary storage device. Data in a data block may be reread by the requesting process or by other processes as long as the requested data remains valid. Lock fields are used to coordinate multi-thread and multi-user accesses.

Current US Cross Reference Classification (3):
711/154

☐ Generate Collection Print

L18: Entry 5 of 13 File: USPT Oct 15, 2002

DOCUMENT-IDENTIFIER: US 6466993 B1
TITLE: Method and apparatus for performing transactions rendering between host processors and I/O devices using concurrent non-blocking queuing techniques and I/O bus write operations

Abstract Text (1):
In a computer system including one or more hosts coupled via a host bus to each other and a cached host memory, an Input/Output processor providing data to peripheral devices and an I/O bus disposed between the hosts and the Input/Output processor for transfer of information therebetween, an inbound queue structure receives message information from one of the hosts, and an outbound queue structure sends message information from the I/O processor to one of the hosts. Each of the queue structures comprises a pair designated as a free-list buffer and a post-list buffer. The free-list buffer of the inbound queue structure and the post-list buffer of the outbound queue structure are locally coupled to the hosts so that message information transfers between these two buffers and the hosts without incurring I/O bus read operations.

Current US Cross Reference Classification (2):
710/310

Current US Cross Reference Classification (3):
710/52

First Hit Fwd Refs

☐ Generate Collection Print

L18: Entry 9 of 13 File: USPT Oct 5, 1999

DOCUMENT-IDENTIFIER: US 5961615 A
TITLE: Method and apparatus for queuing data

Abstract Text (1):
A queue structure includes a plurality of entries, a plurality of ports coupled to the entries, a plurality of enable lines coupled to the entries and the ports, and control logic. Each enable line is adapted to enable a selected port to communicate with a selected entry. The control logic is adapted to enable at least two enable lines and allow at least one of the ports to communicate with at least two of the entries concurrently. A method for storing data in a queue is provided. The queue includes a plurality of entries, a plurality of ports coupled to the entries, and a plurality of enable lines coupled to the entries and the ports. Each enable line is adapted to enable a selected port to communicate with a selected entry. The method includes receiving a first instruction on one of the ports. A first enable line is enabled to allow the port to communicate with a first entry. The first instruction is stored in the first entry. A second enable line is enabled concurrent with enabling the first enable line to allow the port to communicate with a second entry. The first instruction is stored in the second entry.

Current US Original Classification (1):
710/54